Stratus: A Hardware/Software Infrastructure for Controlled Cloud Research

Lucia Pons*, Salvador Petit*, Julio Pons*, María E. Gómez*, Chaoyi Huang† and Julio Sahuquillo*

* Universitat Politècnica de València, Spain
{lupones,spetit,jpons,megomez, jsahuqui}@disca.upv.es

† Huawei Technologies Co. Ltd., China
joehuang@huawei.com

***Abstract*—Cloud systems deploy a wide variety of shared resources and host a large number of tenant applications. To perform cloud research, a small experimental platform is commonly used, which hides the huge system complexity and provides flexibility. Despite being simpler, this platform should include the main cloud system components (hardware and software) to provide representative results. A wide set of platforms have spread in recent years; however, most of them only include a major cloud component or lack the deployment of virtual machines (VMs) to provide isolation.**

This paper presents Stratus, an experimental platform that is currently being used to carry out cloud research. To the best of our knowledge, Stratus is the only platform that jointly provides three main features: uses VMs to isolate tenant applications, deploys the three types of cloud nodes (server, client, and storage), and manages all main shared system resources (CPUs, LLC space, memory, network, and disk bandwidth). Moreover, Stratus implements a software manager to ease the research and aid the design of QoS-aware policies. The manager integrates three main functionalities: management and control of the execution of VMs and running applications, monitoring of hardware performance counters and system resource utilization, and partitioning of the main shared system resources by using technologies available in commercial processors.

***Index Terms*—cloud computing, shared resource management, resource monitoring, resource partitioning, virtualization, experimental framework**

I. Introduction

Cloud platforms allocate a wide variety of tenant applications, so they need to provide high computing and storage capabilities. To this end, and as the cloud evolves with time, cloud platforms consist of a variety of computing nodes, each composed of a set of resources. To improve performance and reduce costs, most of these resources (e.g., cores, main memory, and storage) are shared among tenant applications. To provide isolation and privacy to cloud users, tenant applications are hosted by virtual machines (VMs) [1] in the public cloud. These features make cloud platforms complex systems to deploy, not only from a hardware point of view but also from the software stack perspective, since it needs to support the virtualization of the real hardware. Additionally, it should provide functionalities to meet cloud system requirements such as resource efficiency, SLA (Service of Level Agreement) [2] compliance, and multi-tenancy.

To conduct research in cloud platforms, many companies develop a *small* experimental platform to reduce complexity and provide flexibility, as the experimental workload is under control. For instance, it allows checking if SLA could be violated when applying a resource-management policy and drawing conclusions before implementing it in the real platform. Developing such a controlled experimental platform, however, is challenging since it should be able to provide representative results. To this end, an experimental platform should provide three main features: i) include the main type of hardware nodes (server, client, and storage), ii) provide isolation with VMs to tenant applications, and iii) provide the capabilities to partition all the major shared resources (e.g., main memory, last level cache, network, ...). An important set of platforms or experimental testbeds have spread in recent years to carry out cloud research. However, most platforms used in existing works fail in that they are composed of a single machine [3]–[7], do not provide virtualization [3]–[5], or do not consider the management of important components such as the network [3]–[5] and the remote storage [4], [5], [8]–[10].

This paper presents Stratus, an experimental platform that fulfills the three aforementioned features and that is currently being used to carry out cloud research [11], [12]. Regarding the hardware infrastructure, Stratus includes the three main types of nodes: a server node hosting the tenant applications, a client node launching requests to the server, and a storage node. In this way, network latencies are taken into account. Regarding the software infrastructure, that is, the software stack, we implemented the full software stack (e.g., QEMU, Libvirt, ...) to provide isolation to tenant applications by using VMs. Finally, when purchasing the experimental machines, we checked they were equipped with recent technologies that allow partitioning the last level cache (LLC), memory bandwidth, and so on. In this way, Stratus is allowed to partition the main shared resources.

An important component of Stratus is the manager that integrates three main functionalities: management and control of the execution of VMs and running applications, monitoring of hardware performance counters and system resource utilization, and partitioning the main shared system resources using the technologies available in experimental machines.

Finally, Stratus supports the execution of both client-server workloads (e.g., TailBench [13], CloudSuite [14]) as well as best-effort or batch workloads (e.g., stressor microbenchmarks

[15], [16] or SPEC CPU workloads [17]).

The remainder of this work is organized as follows. Section II compares other platforms used in previous works against Stratus. Section III presents an overview of Stratus' experimental platform considering, both the deployed hardware and system software. Section IV describes Stratus' resource and application manager and summarizes the technologies used in Stratus in order to monitor and partition the main shared system resources. In Section V some experimental results are presented for illustrative purposes and finally, some conclusions are drawn in Section VI.

II. Existing Solutions

As cloud infrastructures evolve with time, the number and variety of tenant applications as well as the available tools are continuously increasing. This situation has prompted researchers to create new testbeds to conduct their research.

At a large scale, testbeds such as Grid'500 [19], Cloudlab [20], and Chamaleon [21] have been built to allow researchers to carry out experiments on distributed systems deployed in multiple sites spread geographically. Under these testbeds, users request the desired resources for a limited amount of time using a reservation system, and then, configure such resources for their use (e.g., deploy a custom software stack).

This paper focuses on built-in experimental testbeds for small-scale research that users deploy in their research facilities without relying on external systems. Table I summarizes, for a representative subset of testbeds recently appeared, how they fulfill the three aforementioned features: i) if they support virtualization with VMs, ii) the type of nodes the platform includes, and resources that can be monitored or managed. For comparison purposes, the bottom row of the table includes our experimental platform, Stratus. The table shows that, to the best of our knowledge, there is no existing work that has made use of a controlled experimental platform that includes all the features included in Stratus.

Stratus deploys the three types of nodes, uses VMs to allocate tenant applications, and provides monitoring and partitioning capabilities of the main system resources. As it can be seen in the table, only the testbed used to evaluate Skynet [18] includes the three types of nodes that Stratus deploys. Some approaches [3] run server and clients on the same machine or use single-node experimental platforms [4], [5], obviating the network interference.

Regarding virtualization, only three of the listed works use VMs to contain the tenant applications. Two of these approaches [3], [9] also use Linux KVM to deploy VMs. Some approaches use containers, which allow better performance at the expense of worse isolation. Software isolation tools (e.g., cgroups) indeed enable resource isolation in containers. However, containers are forced to use the same kernel as the host; thus, isolation is not possible at the kernel level.

Concerning the management of the shared resources, only PARTIES [8] takes into consideration all the shared resources similar to Stratus, but notice that this platform lacks a storage node and makes use of containers instead of VMs. From the listed resources, the CPU (which embraces hardware performance counters and CPU utilization monitoring, as well as allocation of CPU cores) is the only resource considered in all of the works. On the other hand, the network and disk are the shared resources least considered.

Apart from the platforms analyzed in Table I, other research works make use of experimental platforms that focus on a single specific shared resource. Less Provisioning [22] and Twig [7] focus on CPU resource allocation by dynamically adjusting the CPU resources based on resource utilization and hardware performance counters, respectively. ReTail [6] also focuses on CPU resources but manages these resources by adjusting the CPU frequency. QWin [23] was devised to guarantee tail latency SLO of distributed storage servers by partitioning cores of storage servers among tenants applications. Finally, LIBRA [24] proposes a framework for dynamic memory bandwidth management.

III. Overview of Stratus' Experimental Framework

Figure 1 presents an overview of Stratus' experimental framework. The framework is made up of three main nodes: server, client, and storage. The server node acts as the server side in our client-server architecture. This node runs the VMs hosting the server applications, which are managed by Stratus' resource and application manager software described in Section IV. The server node is interconnected to the client and storage nodes with two distinct 20 Gbps networks. The client node is an auxiliary node that emulates client behavior by executing client applications that perform requests to the VMs in the server node. Finally, the storage node provides remote storage resources for the VMs.

The design choices to set up Stratus' experimental platform are taken in two main axes: the deployed hardware and the system software. This section presents and motivates the choices we selected for each axis.

A. Deployed Hardware

Types and Number of Nodes. A key design decision is selecting the node types and the amount of them in the experimental framework. On the one hand, the huge complexity of managing a high number of machines that are found in real environments should be avoided. On the other hand, the results provided by the framework must be representative of real scenarios. For this purpose, the number of nodes is minimized while keeping at least one node for each of the existing actors in real cloud environments.

The first step when selecting the number of nodes is to analyze which actors are typically present in cloud systems. A cloud system includes two main types of nodes: computing nodes and storage nodes. Thus, a minimal experimental framework requires at least one node of each of these types. In addition, a separate node is needed to emulate client behavior.

Hardware Specifications. Regarding server and storage nodes, they should be representative of typical nodes in cloud infrastructures. This implies appropriate specifications for the

TABLE I: Summary of experimental infrastructure used in general resource-oriented works.

Paper	Year	VMs	Type of Node			Resource Monitoring/Partitioning				
			Server	Client	Storage	CPU	LLC	Mem.BW	Disk	Net.
ServerMore [3]	2021	✓	✓	✗	✓	✓	✓	✓	✗	✗
Skynet [18]	2021	✗, containers	✓	✓	✓	✓	✗	✗	✓	✓
Alita [4]	2020	✓	✓	✗	✗	✓	✓	✓	✗	✗
CLITE [5]	2020	✗	✓	✗	✗	✓	✓	✓	✓	✗
PARTIES [8]	2019	✗, containers	✓	✓	✗	✓	✓	✓	✓	✓
Scavenger [9]	2018	✓	✓	✓	✗	✓	✓	✗	✗	✓
Vertical Elasticity [10]	2018	✗, containers	✓	✓	✗	✓	✓	✗	✓	✓
Stratus	2022	✓	✓	✓	✓	✓	✓	✓	✓	✓

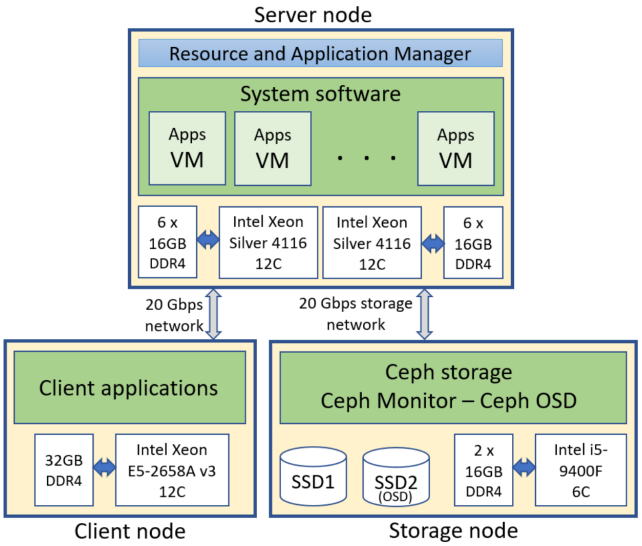


Fig. 1: Overview of Stratus' experimental framework.

TABLE II: Node hardware specifications (processor package and main memory).

Node	Processor Package		Main Memory
	Processor	#Cores (#Threads)	
Main	2x Intel Xeon Silver 4116	48 (96)	12 x DDR4-2666 16GB DIMMs
Client	Intel E5-2658A	12 (24)	1 x 32GB DIMM
Storage	Intel i5-9400F	6 (6)	2 x DDR4-2666 16GB DIMMs

processor package (like the number and type of cores) as well as main memory features (e.g., number of channels, DIMMs, and storage capacities). In the case of the storage node, the persistent solid state (SSD) or hard disk (HDD) drives must also be taken into account.

Table II shows the specifications (processor package and main memory) for each of the nodes. The specifications of the server and storage nodes can be considered representative of the nodes implemented in real cloud systems. For example, Google Cloud CPU platforms [25], Amazon EC2 C6 and C5 instances [26] and Huawei Elastic Cloud servers [27] use Intel Xeon Scalable Processors including from tens to hundreds of main memory capacity.

Regarding the storage media in the storage node (shown in Figure 1), it is composed of two SSD devices. The first one (SSD1) contains the storage node OS and system software while the second one (SSD2) with 960GB is exclusively devoted to acting as remote persistent storage for server applications running in the server node VMs. This persistent storage is served to the **server** node as a *Ceph Object Storage Device (OSD)*. Ceph [28] is an open-source distributed storage platform that is commonly installed in cloud environments.

Finally, the client and storage nodes are interconnected to the server node with two dedicated 20 Gbps networks. More specifically, the server node has two 20 Gbps network cards (dual port, 10 Gbps per port) that connect to the client and storage nodes.

Note that, as in real cloud systems, the actual hardware specifications of Stratus' experimental hardware can be replaced and upgraded as technology, the market, and applications' requirements evolve.

B. System Software

To build our experimental framework we analyzed the main components of OpenStack [29]. OpenStack is an open-source cloud computing software stack and it is a *de facto* standard for the management of virtual services in both public and private clouds. OpenStack is a complex software framework, with multiple components and add-ons supporting different types of hardware devices (e.g., network devices, storage devices, etc.) from multiple vendors. To avoid dealing with such complexity, we built a simpler framework that includes the main software components that can be found in a typical OpenStack deployment. The major simplification lies on the top software levels, which aim to reduce management complexity but have a negligible or null impact on performance monitoring and partitioning, which is the main purpose of Stratus' framework.

Figure 2 presents the main components of the system software deployed in the server node and their interactions. These components, which manage the VMs and the network interconnections, are described below.

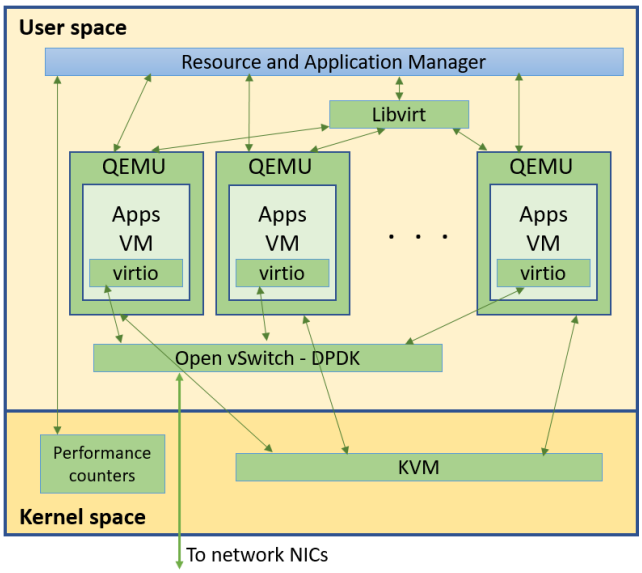


Fig. 2: Stratus' system software deployed in the server node.

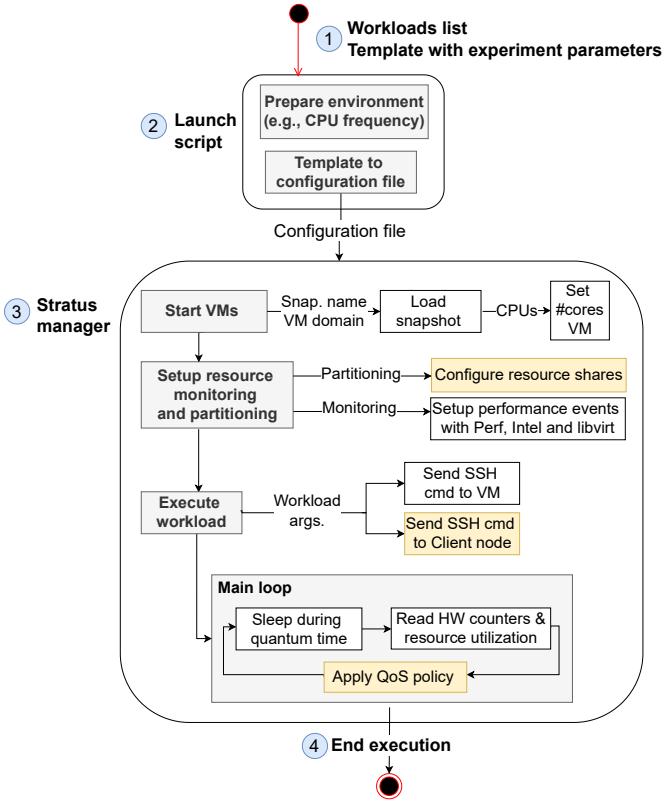


Fig. 3: Workflow followed by Stratus' resource and application manager to launch experiments. Yellow boxes represent actions that are optional.

VM Infrastructure. The execution and management of VMs involve a complex software stack, where three main levels can be distinguished: the hypervisor, the virtual resource manager, and the guest OS and applications. The hypervisor refers to the OS installed in the host physical machine (PM). A wide set of both open-source and proprietaries hypervisors are being used currently in the industry. Examples of open-source hypervisors are Linux with KVM [30] and Xen [31]. The former is one of the current industry trends, and it is being used by Amazon [32] and Google [33]. The latter is also supported by Amazon. The virtualization manager refers to the software platform that manages the PM hardware resources and distributes them among VMs. One example of a virtualization manager is Libvirt [34]. Some virtualizers, like QEMU [35], also support both KVM and Xen. Finally, the guest OS and tenant applications run in the different VMs. Guest OS and applications can be either proprietary or open source (e.g., a Linux server distribution executing several Internet services).

Network Software. To interconnect the VMs with the physical network interface cards (NICs) of the server node, a *virtual switch* is used. The virtual switch is set up with Open vSwitch (OvS) [36]. Emulated NICs at the VMs (i.e., virtio [37] NICs) and each physical NIC (both ports) in the server node are accessed from the virtual switch through the Data Plane Development Kit (DPDK) [38]. DPDK enables the direct transfer of packets between virtio NICs and physical NICs, bypassing the host OS kernel network stack. This setup boosts network performance compared to the default packet forwarding mechanism implemented in the Linux kernel.

IV. Stratus' Resource and Application Manager

Providing an experimental framework that allows automating the setup and execution of experiments is crucial when carrying out research. The goal of Stratus' resource and application manager is to assist the researcher in this task, providing a friendly interface. To this end, it implements three main functions: i) to manage and control the execution of one or more VMs, each one running an application, ii) to monitor hardware performance counters and system resource utilization, iii) to partition system resources and assign them to VMs.

A. Execution of Experiments

To illustrate how Stratus' resource and application manager performs experiments, Figure 3 shows a block diagram of the workflow when carrying out experiments using Stratus's manager. The diagram illustrates the main steps performed when launching one or more VMs together with the applications to be run on them (*VM-application pairs*). Next, each of the steps is discussed in detail.

1) Define experiment workload and parameters: As a prior step, the workload (i.e., VMs and applications to be run on them) and experimental conditions must be defined. To ease this task, Stratus makes use of MAKO templates [39], which provides a simple and intuitive language to specify the parameters of the experiments: VMs and applications to be executed (domain name, workload, number of CPUs, etc.), VCPUs core pinning, performance events to be monitored,

length of the quantum, etc. The template can also specify if VMs are only allowed to use a partition of a shared resource (LLC, memory bandwidth, network bandwidth, or disk bandwidth).

2) Execute Launch script to start the manager: To start running an experiment, the user executes the *launch* script. First, the script prepares the execution environment. For instance, fixing the processor frequency to avoid variability among experiments. Additionally, the server clocks of both the server and client machines are synchronized to ensure server- and client-collected metrics are aligned, using the Network Time Protocol (NTP) [40] with a known NTP time server (europe.pool.ntp.org). When the environment is ready, the configuration file is generated with the workloads to execute and all the experiment parameters from the MAKO template. Then, the manager starts to run.

3) Prepare VMs for execution: The first step the manager performs is setting up and starting the VMs. To reduce the start-up overhead, the manager makes use of the *snapshots* feature of Libvirt. A snapshot is a copy of the state of a VM, including the disk and main memory contents. This feature preserves a VM's actual state and data at a given time. Therefore, this state can be reverted at any moment. For each VM, we have taken a snapshot that has already performed the OS boot process and is ready to receive the command to launch the target benchmark. Once the VMs are started, and the snapshots are loaded, the number of CPUs of each VM (i.e., VCPUs) can be modified in case a multi-threaded application is going to be executed and more than one CPU is required.

4) Setup resource monitoring and partitioning: With QEMU, each VCPU is associated with a processor ID (PID) in the host OS. These PIDs are required to monitor hardware performance counters with Perf individually for each thread (i.e., VCPU) of the VM. Similarly, LLC and memory bandwidth monitoring is performed on a PID basis. The remaining resources, network, and disk bandwidth are monitored per VM. The manager also allows partitioning of the main system shared resources and assigns each VM a share of a given resource. Therefore, if specified in the template, a resource share is allocated to the VM.

5) Start running applications in the VMs: When the VMs are operative and ready to start executing the applications, an SSH command is sent to each VM to start the execution of each workload. Stratus' manager is adapted to support the execution of client-server workloads (e.g., TailBench benchmark suite) as well as best-effort or batch workloads (e.g., stressor microbenchmarks or SPEC CPU benchmarks). In the case of a client-server workload, an SSH command is sent to the client node to start running the clients, which send requests to the server (already running).

6) Perform actions in each quantum: Once the execution starts, the manager executes the *main loop* (see Figure 3) for the rest of the execution time. In each iteration, the manager is suspended for a given quantum length (established in the template). Then, data is collected from different sources (e.g., hardware performance counters, Linux file system, Intel library, or Libvirt) to monitor the main system resources (CPU usage, LLC occupancy, memory, network, and disk bandwidth). Additionally, the manager is adapted to allow implementing and applying QoS policies. For instance, policies that manage resource sharing among VMs [4], [8], [9], predict interference among VMs [5], [6], [10], [12], [41] or schedule VMs [42].

7) Execution end: The main loop ends when the manager detects that all the VMs have finished running their applications, the moment at which it shuts down the running VMs.

All the data collected from the hardware performance counters and system resources are stored in CSV files, ready to be processed. Additionally, for characterization and debugging purposes, statistics and data are also collected inside the VMs. For instance, in Tailbench workloads, the clients report results such as latency per query, queries per interval, tail latency, etc.

B. Monitoring & Partitioning Main Shared Resources

Tenant applications compete for shared resources in cloud systems. This means that the performance of a given tenant application (or VM) will depend on the co-running applications. In other words, on the share of the resource that is able to use. As a consequence, it is worth studying to which extent the performance of a given application is affected by varying the amount of share allocated to the application.

To perform this kind of experiments, and help researchers in their decision-making, we need to define the tools to be implemented. In the last few years, server processors have been provided with advanced technologies that allow monitoring and partitioning of the major system resources.

Below, we explain how monitoring and partitioning of each shared resource is implemented in Stratus without relying on any external tool.

CPU Utilization. CPU utilization accounts for the percentage of time a CPU is active. It is a crucial metric in cloud environments since CPU utilization has been proven to be low (less than 20%) most of the time [43], [44], and thus, many resource provisioning strategies [9], [22], [45] seek to improve the CPU usage. To obtain the utilization of each CPU, we use the data collected from the file `/proc/stat`, which reports statistics about the kernel activity aggregated since the system first booted. To pin the VMs' VCPUs to logical cores of the physical machine, Stratus uses Libvirt's API [34].

Last Level Cache (LLC). Due to the high latency to access to main memory upon LLC misses, the LLC is one of the *critical* shared resources in current multi-core processors. Recently, some processor manufacturers like Intel have developed technologies that allow monitoring and partitioning of the LLC. In Intel processors, these technologies are known as Cache Monitoring Technology (CMT) and Cache Allocation Technology (CAT) [46]. Partitioning is performed using Classes of Service (CLOS), which can be defined either as groups of applications (PIDs) or as groups of logical cores to which a partition of the LLC is assigned. The LLC is

partitioned in a *per way* basis, that is, a cache way acts as the granularity allocated to CLOS.

Memory Bandwidth. Memory bandwidth can considerably impact the performance or responsiveness of applications. For instance, in a server system with different VMs accessing the main memory, the inter-VM interference can significantly grow and make the most memory-sensitive VMs perform below an acceptable level, compromising the QoS. Recent Intel Xeon Scalable processors introduce Memory Bandwidth Allocation (MBA) [47], which allows to distribute memory bandwidth between the running applications. More precisely, it allows controlling the memory bandwidth between the L2 and the L3 (i.e., LLC) caches. Similarly to CAT, MBA works using CLOS. That is, MBA bandwidth limits apply only to CLOS, to which the user can assign tasks (PIDs) or cores. However, MBA **works on a per-core basis**. If the individual memory bandwidths of two applications running on the same core are limited with different values, the *maximum* limitation is the one that will apply to that core.

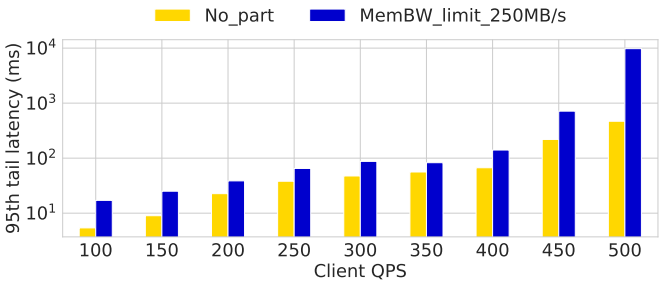
Disk Bandwidth. Many workloads operate on big data files or databases that cannot be completely loaded to main memory. Consequently, these workloads need to constantly rely on the I/O system to access the disk and load/store the required data. Monitoring and partitioning this subsystem is, therefore of high interest. I/O access to the disks can be monitored using the virsh tool or Libvirt's API. Both mechanisms offer the same functionality and allow monitoring the number of read, write, and flush operations, the number of bytes read and written, as well as the total duration of the read, write, and flush operations.

Network Bandwidth. VMs running on the same physical machine share network resources whose bandwidth and latency play an important role in the QoS of tenant applications. Consequently, network resources should be monitored and partitioned to minimize inter-VM interference. The number of network packets or bytes that go through a network interface can be monitored with Libvirt's API.

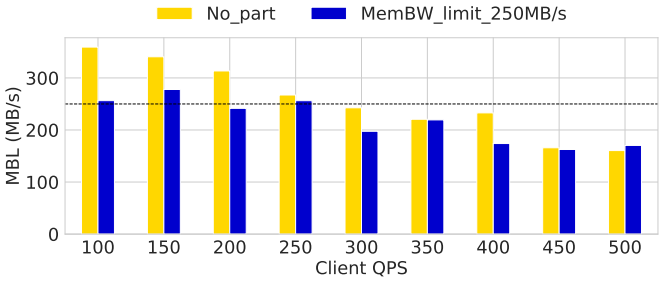
V. Case Studies: Latency-Critical Workloads

Latency-critical applications are increasingly common in data centers. These applications typically support online interactive services (e.g., web search) and must respond to the input requests within certain latency bounds to guarantee QoS (e.g., the 95^{th} or 99^{th} percentile latency) and provide a satisfactory user experience. Cloud benchmark suites [13], [14], [48] have been designed to include representative applications of today's latency-critical applications. Among these, the TailBench benchmark suite [13] offers a set of representative latency-critical applications that can be easily configured (e.g., number of server threads, client requests) and report results from the client-side (e.g., 95^{th} tail latency).

This section presents, for illustrative purposes, some experimental results obtained with the `xapian` application from the TailBench suite to show the experimental capabilities of Stratus. `Xapian` [49] is an open-source online search engine that is widely used on many popular websites. The benchmark



(a) Latency



(b) Memory bandwidth

Fig. 4: Response latency and memory bandwidth of the `xapian` server when limiting the memory bandwidth to 250MB/s

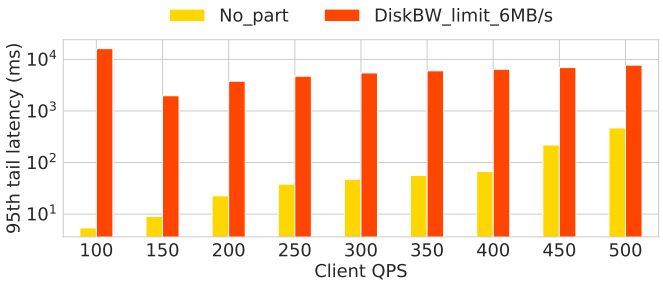
is configured so that it represents a leaf node in a search engine populated with the English version of Wikipedia. We have configured the server to launch `xapian` in a VM with 1 VCPU. The number of requests of the experiment has been set to 10000, and an additional 1000 requests are used to warm up the server. To explore the server performance under different load levels, the clients have been set to generate a number of queries per second (QPS) ranging from 100 to 500 in steps of 50.

A. Memory Bandwidth Allocation

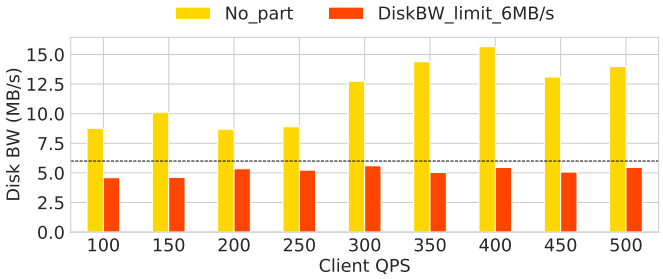
This case study discusses the functionality of partitioning the memory bandwidth implemented in Stratus using Intel MBA technology (see Section IV-B). Unlike LLC partitioning, memory bandwidth allocation has been less explored in existing solutions even though many popular applications in today's data centers (big data, graph processing) make intensive use of this resource.

Figure 4 shows the results of `xapian` executed under no restrictions (`No_part`, yellow bars) and the results when the memory bandwidth (blue bars) is limited to 250MB/s for different values of client QPS. Figure 4a shows the 95^{th} percentile latency and Figure 4b shows the consumed memory bandwidth (the dashed line shows the imposed memory bandwidth limit).

As it can be observed in Figure 4b, `xapian` shows a low memory bandwidth consumption (at most 350 MB/s) compared to the maximum peak bandwidth achievable per core (around 9-10 GB/S). Despite this, results show that reducing



(a) Latency



(b) Disk bandwidth

Fig. 5: Response latency and disk I/O bandwidth of the `xapian` server when limiting the disk overall bandwidth of the VM to 6MB/s.

the available memory bandwidth impacts the response latency. An interesting observation is that tail latency grows most in experiments with a high QPS value which presents the lowest bandwidth consumption. However, notice that results reported in this graph represent average values, and in experiments with high QPS, more peak values are observed in the memory bandwidth, which is affected by the imposed limit and significantly hurts the response latency.

B. Disk I/O Throttling

Interference in disk bandwidth is one of the least studied system resources in literature. However, the trend of increasing the working set of applications to a point where it does not fit in the main memory will eventually force applications to make use of disk storage.

This case study evaluates the 95^{th} tail latency and the total disk bandwidth (read and write) achieved by `xapian` while varying the client QPS and setting a bandwidth constraint. Figure 5 presents the 95^{th} tail latency and the disk bandwidth achieved by `xapian` varying the QPS from 100 to 500. The figure presents both metrics when the VM runs without any resource constraint (`No_part`, yellow bars) and when the total disk bandwidth is limited to 6MB/s (orange bars).

Results show that the disk bandwidth constraint hurts significantly the performance of `xapian`, increasing the tail latency up to three orders of magnitude. This means that special consideration should be given to contention at the disk bandwidth, as delaying read and/or write operations to disk has proved to have a severe impact on the response latency.

VI. Conclusions

Research in cloud systems is becoming increasingly popular. However, before deploying solutions to public cloud systems, research, and tests should be performed in controlled experimental platforms. Existing solutions make use of testbeds to evaluate their work, but these platforms do not *jointly* deploy the main features of real cloud systems (types of nodes, virtualization, resource management) and thus, do not provide representative results.

This paper presents Stratus, an experimental platform used to carry out cloud research and controlled experiments. Unlike experimental setups used in existing work, Stratus complies with all the features of cloud environments in terms of hardware and software deployment. Additionally, Stratus implements an application and resource manager that assists the researcher in the execution of experiments and resource management, which is key to designing QoS policies to mitigate inter-VM interference.

VII. Acknowledgments

This work has been partially supported by Huawei Cloud, by the Spanish Ministerio de Universidades under the grant FPU18/01948, and by the Spanish Ministerio de Ciencia e Innovación and European ERDF under grants PID2021-123627OB-C51 and TED2021-130233B-C32, and by Generalitat Valenciana under Grant AICO/2021/266.

References

[1] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *2010 Second International Conference on Computer and Network Technology*, 2010, pp. 222–226.

[2] D. Serrano, S. Bouchenak, Y. Kouki, F. A. de Oliveira Jr., T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "Sla guarantees for cloud services," *Future Generation Computer Systems*, vol. 54, pp. 233–246, 2016.

[3] A. Suresh and A. Gandhi, "Servermore: Opportunistic execution of serverless functions in the cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21, 2021, p. 570–584.

[4] Q. Chen, S. Xue, S. Zhao, S. Chen, Y. Wu, Y. Xu, Z. Song, T. Ma, Y. Yang, and M. Guo, "Alita: Comprehensive performance isolation through bias resource management for public clouds," in *Proceedings of SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–13.

[5] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 193–206.

[6] S. Chen, A. Jin, C. Delimitrou, and J. F. Martínez, "Retail: Opting for learning simplicity to enable qos-aware power management in the cloud," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 155–168.

[7] R. Nishtala, V. Petrucci, P. Carpenter, and M. Sjalander, "Twig: Multi-agent task management for colocated latency-critical cloud services," in *Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 167–179.

[8] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr. 2019, pp. 107–120.

[9] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2019, p. 272–285.

[10] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, and X. Koutsoukos, "Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications," in *Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD)*, Jul. 2018, pp. 82–89.

[11] L. Pons, J. Feliu, J. Puche, C. Huang, S. Petit, J. Pons, M. E. Gómez, and J. Sahuquillo, "Effect of hyper-threading in latency-critical multithreaded cloud applications and utilization analysis of the major system resources," *Future Generation Computer Systems*, vol. 131, pp. 194–208, 2022.

[12] L. Pons, J. Feliu, J. Sahuquillo, M. E. Gómez, S. Petit, J. Pons, and C. Huang, "Cloud white: Detecting and estimating qos degradation of latency-critical workloads in the public cloud," *Future Generation Computer Systems*, vol. 138, pp. 13–25, 2023.

[13] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[14] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[15] Canonical Ltd, "Ubuntu manpage: stress-ng," Available at https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html, 2020, accessed: 2022-11-20.

[16] ESnet, NLANR, DAST, "iperf tool for network bandwidth measurements," Available at https://iperf.fr/, 2020, accessed: 2022-11-20.

[17] "Why would a cloud computing company use the spec cpu2017 benchmark suite?" Available at https://www.spec.org/cpu2017/publications/DO-case-study.html, 2017, accessed: 2019-08-02.

[18] Y. Sfakianakis, M. Marazakis, and A. Bilas, "Skynet: Performance-driven resource management for dynamic workloads," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 527–539.

[19] S. Badia, A. Carpen-Amarie, A. Lèbre, and L. Nussbaum, "Enabling large-scale testing of iaas cloud platforms on the grid'5000 testbed," in *Proceedings of the 2013 International Workshop on Testing the Cloud*, 2013, pp. 7–12.

[20] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab." in *USENIX Annual Technical Conference*, 2019, pp. 1–14.

[21] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock *et al.*, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, 2020, pp. 219–233.

[22] B. Cai, K. Li, L. Zhao, and R. Zhang, "Less provisioning: A hybrid resource scaling engine for long-running services with tail latency guarantees," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1941–1957, 2022.

[23] L. Ma, Z. Liu, J. Xiong, and D. Jiang, "Qwin: Core allocation for enforcing differentiated tail latency slos at shared storage backend," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 1098–1109.

[24] Y. Zhang, J. Chen, X. Jiang, Q. Liu, I. M. Steiner, A. J. Herdrich, K. Shu, R. Das, L. Cui, and L. Jiang, "Libra: Clearing the cloud through dynamic memory bandwidth management," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 815–826.

[25] "Google cloud compute engine - cpu platforms [online]," Available at https://cloud.google.com/compute/docs/cpu-platforms, 2022, accessed: 2022-11-14.

[26] "Amazon's ec2 [online]," Available at https://aws.amazon.com/ec2/instance-types/?nc1=h_ls, 2022, accessed: 2022-11-14.

[27] "Huawei elastic cloud server (ecs) [online]," Available at https://www.huaweicloud.com/intl/en-us/product/ecs.html, 2022, accessed: 2022-11-14.

[28] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system." in *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI)*, Nov. 2006, pp. 307–320.

[29] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: Toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, Mar. 2012.

[30] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, no. 8. Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[31] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 164–177, 2003.

[32] "Amazon web services [online]," Available at https://aws.amazon.com/ec2/faqs/?nc1=h_ls, 2022, accessed: 2022-11-28.

[33] "Google compute engine faq [online]," Available at https://cloud.google.com/compute/docs/faq, 2022, accessed: 2022-11-28.

[34] "ibvirt: The virtualization api [online]," Available at https://libvirt.org, 2022, accessed: 2022-11-28.

[35] "Qemu [online]," Available at https://www.qemu.org, 2022, accessed: 2022-11-28.

[36] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[37] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[38] "Dpdk [online]," Available at https://www.dpdk.org/, 2022, accessed: 2022-11-28.

[39] Michael Bayer et al., "Mako Templates," Available at http://www.makotemplates.org/, 2019.

[40] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[41] R. Jia, Y. Yang, J. Grundy, J. Keung, and L. Hao, "A systematic review of scheduling approaches on multi-tenancy cloud platforms," *Information and Software Technology*, vol. 132, p. 106478, 2021.

[42] Z. Wang, C. Xu, K. Agrawal, and J. Li, "Adaptive scheduling of multi-programmed dynamic-multithreading applications," *Journal of Parallel and Distributed Computing*, vol. 162, pp. 76–88, 2022.

[43] C. Lu, K. Ye, G. Xu, C.-Z. Xu, and T. Bai, "Imbalance in the cloud: An analysis on alibaba cluster trace," in *2017 IEEE International Conference on Big Data*, 2017, pp. 2884–2892.

[44] Q. Liu and Z. Yu, "The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2018, p. 347–360.

[45] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, 2017, p. 153–167.

[46] Intel, "Improving real-time performance by utilizing cache allocation technology," *Intel Corporation, April*, 2015.

[47] Andrew H., Abbasi, Khawar M., Marcel C., "Introduction to memory bandwidth allocation," Available at https://software.intel.com/en-us/articles/introduction-to-memory-bandwidth-allocation, 3 2019.

[48] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: A big data benchmark suite from internet services," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 488–499.

[49] "Xapian project [online]," Available at https://github.com/xapian/xapian, 2022, accessed: 2022-11-30.